(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 May 2002 (10.05.2002)

PCT

(10) International Publication Number
WO 02/37799 A2

(51) International Patent Classification[7]: H04L 29/06

(21) International Application Number: PCT/US01/47013

(22) International Filing Date:
5 November 2001 (05.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/245,788	3 November 2000 (03.11.2000)	US
60/245,789	3 November 2000 (03.11.2000)	US
60/245,790	3 November 2000 (03.11.2000)	US
60/245,859	3 November 2000 (03.11.2000)	US
09/878,787	11 June 2001 (11.06.2001)	US
09/930,014	15 August 2001 (15.08.2001)	US
09/965,526	26 September 2001 (26.09.2001)	US

(71) Applicant: **THE BOARD OF REGENTS OF THE UNIVERSITY OF NEBRASKA** [US/US]; 3835 Holdrege, Lincoln, NE 68588-0745 (US).

(72) Inventor: **GODDARD, Steve**; The Board of Regents of the University of Nebraska, 3835 Holdrege Street, Lincoln, NE 68588-0745 (US).

(74) Agents: **THOMAS, Michael, J.** et al.; Senniger, Powers, Leavitt & Roedel, One Metropolitan Square, 16th Floor, St. Louis, MO 63102 (US).

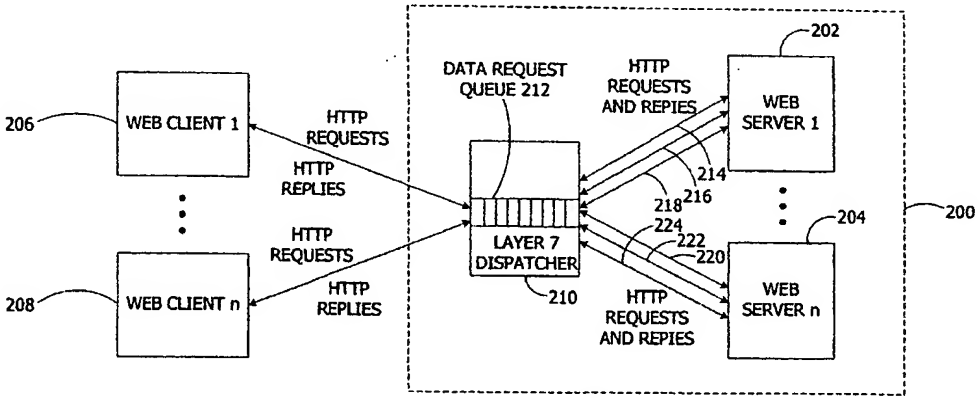
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: CONTROLLED SERVER LOADING



(57) Abstract: Standalone and cluster-based servers, including Web servers, control the amount of data processed concurrently by such servers to thereby control server operating performance. A dispatcher is preferably interposed between clients and one or more back-end servers, and preferably monitors the performance of each back-end server (either directly or otherwise). For each back-end server, the dispatcher preferably also controls, in response to the monitored performance, either or both the number of concurrently processed data requests and the number of concurrently supported connections to thereby control the back-end servers' performance. In one embodiment, the dispatcher uses a packet capture library for capturing packets at OSI layer 2 and implements a simplified TCP/IP protocol in user-space (vs. kernel space) to reduce data copying. Commercially off-the-shelf (COTS) hardware and operating system software are preferably employed to take advantage of their price-to-performance ratio.

WO 02/37799 A2

CONTROLLED SERVER LOADING

Field of the Invention

[0001] The present invention relates generally to controlled loading of servers, including standalone and cluster-based Web servers, to thereby increase server performance. More particularly, the invention relates to methods for controlling the amount of data processed concurrently by such servers, including the number of connections supported, as well as to servers and server software embodying such methods.

Background of the Invention

[0002] A variety of Web servers are known in the art for serving the needs of the over 100 million Internet users. Most of these Web servers provide an upper bound on the number of concurrent connections they support. For instance, a particular Web server may support a maximum of 256 concurrent connections. Thus, if such a server is supporting 255 concurrent connections when a new connection request is received, the new request will typically be granted. Furthermore, most servers attempt to process all data requests received over such connections (or as many as possible) simultaneously. In the case of HTTP/1.0 connections, where only one data request is associated with each connection, a server supporting a maximum of 256 concurrent connections may attempt to process as many as 256 data requests simultaneously. In the case of HTTP/1.1 connections, where multiple data requests per connection are permitted, such a server may attempt to process in excess of 256 data requests concurrently.

[0003] The same is true for most cluster-based Web servers, where a pool of servers are tied together to act as a single

unit, typically in conjunction with a dispatcher that shares or balances the load across the server pool. Each server in the pool (also referred to as a back-end server) typically supports some maximum number of concurrent connections, which may be the same as or different than the maximum number of connections supported by other servers in the pool. Thus, each back-end server may continue to establish additional connections (with the dispatcher or with clients directly, depending on the implementation) upon request until its maximum number of connections is reached.

[0004] The operating performance of a server at any given time is a function of, among other things, the amount of data processed concurrently by the server, including the number of connections supported and the number of data requests serviced. As recognized by the inventor hereof, what is needed is a means for dynamically managing the number of connections supported concurrently by a particular server, and/or the number of data requests processed concurrently, in such a manner as to improve the operating performance of the server.

[0005] Additionally, most cluster-based servers that act as relaying front-ends (where a dispatcher accepts each client request as its own and then forwards it to one of the servers in the pool) create and destroy connections between the dispatcher and back-end servers as connections between the dispatcher and clients are established and destroyed. That is, the state of the art is to maintain a one-to-one mapping of back-end connections to front-end connections. As recognized by the inventor hereof, however, this can create needless server overhead, especially for short TCP connections including those common to HTTP/1.0.

Summary of the Invention

[0007] In order to solve these and other needs in the art, the inventor has succeeded at designing standalone and cluster-based servers, including Web servers, which control the amount of data processed concurrently by such servers to thereby control server operating performance. As recognized by the inventor, it is often possible to increase one or more performance metrics for a server (e.g., server throughput) by decreasing the number of concurrently processed data requests and/or the number of concurrently supported connections. A dispatcher is preferably interposed between clients and one or more back-end servers, and preferably monitors the performance of each back-end server (either directly or otherwise). For each back-end server, the dispatcher preferably also controls, in response to the monitored performance, either or both of the number of concurrently processed data requests and the number of concurrently supported connections to thereby control the back-end servers' performance. In one embodiment, the dispatcher uses a packet capture library for capturing packets at OSI layer 2 and implements a simplified TCP/IP protocol in user-space (vs. kernel space) to reduce data copying. Commercially off-the-shelf (COTS) hardware and operating system software are preferably employed to take advantage of their price-to-performance ratio.

[0008] In accordance with one aspect of the present invention, a server for providing data to clients includes a dispatcher having a queue for storing requests received from clients, and at least one back-end server. The dispatcher stores in the queue one or more of the requests received from clients when the back-end server is unavailable to process the one or more requests. The dispatcher retrieves the one or more requests from the queue for forwarding to the back-end server when the back-end server becomes available to process

them. The dispatcher determines whether the back-end server is available to process the one or more requests by comparing a number of connections concurrently supported by the back-end server to a maximum number of concurrent connections that the back-end server is permitted to support, where the maximum number is less than a maximum number of connections which the back-end server is capable of supporting concurrently.

[0009] In accordance with another aspect of the present invention, a method for controlled server loading includes the steps of defining a maximum number of concurrent connections that a server is permitted to support, limiting a number of concurrent connections supported by the server to the maximum number, monitoring the server's performance while it supports the concurrent connections, and dynamically adjusting the maximum number as a function of the server's performance to thereby control a performance factor for the server.

[0010] In accordance with a further aspect of the present invention, a method for controlled server loading includes the steps of receiving a plurality of data requests from clients, forwarding a number of the data requests to a server for processing, and storing at least one of the data requests until the server completes processing at least one of the forwarded data requests.

[0011] In accordance with still another aspect of the present invention, a method for controlled server loading includes the steps of defining a maximum number of data requests that a server is permitted to process concurrently, monitoring the server's performance, and dynamically adjusting the maximum number in response to the monitoring step to thereby adjust the server's performance.

[0012] In accordance with a further aspect of the invention, a method for controlled loading of a cluster-based server

having a dispatcher and a plurality of back-end servers includes the steps of receiving at the dispatcher a plurality of data requests from clients, forwarding a plurality of the data requests to each of the back-end servers for processing, and storing at the dispatcher at least one of the data requests until one of the back-end servers completes processing one of the forwarded data requests.

[0013] In accordance with yet another aspect of the invention, a method for controlled loading of a cluster-based server having a dispatcher and a plurality of back-end servers includes the steps of defining, for each back-end server, a maximum number of data requests that can be processed concurrently, monitoring the performance of each back-end server, and dynamically adjusting the maximum number for at least one of the back-end servers in response to the monitoring step to thereby adjust the performance of the cluster-based server.

[0014] In accordance with still another aspect of the present invention, a server for providing data to clients includes an OSI layer 4 dispatcher having a queue for storing connection requests received from clients, and at least one back-end server. The dispatcher stores in the queue one or more of the connection requests received from clients when the back-end server is unavailable to process the one or more connection requests. The dispatcher retrieves the one or more connection requests from the queue for forwarding to the back-end server when the back-end server becomes available to process the one or more connection requests. The dispatcher also determines whether the back-end server is available to process the one or more connection requests by comparing a number of connections concurrently supported by the back-end server to a maximum number of concurrent connections that the back-end server is permitted to

support, where this maximum number is less than a maximum number of connections which the back-end server is capable of supporting concurrently.

[0001] In accordance with another aspect of the present invention, a method for controlled server loading includes receiving a plurality of connection requests from clients, establishing, in response to some of the connection requests, a number of concurrent connections between a server and clients, and storing at least one of the connection requests until one of the established connections is terminated.

[0002] In accordance with a further aspect of the present invention, a method for controlled server loading includes defining a maximum number of concurrent connections that a server is permitted to support, monitoring the server's performance, and dynamically adjusting the maximum number in response to the monitoring to thereby adjust the server's performance.

[0003] In accordance with still another aspect of the present invention, a method is provided for controlled loading of a cluster-based server having a dispatcher and a plurality of back-end servers. The method includes receiving at the dispatcher a plurality of connection requests from clients, forwarding a plurality of the connection requests to each of the back-end servers, each back-end server establishing a number of concurrent connections with clients in response to the connection requests forwarded thereto, and storing at the dispatcher at least one of the connection requests until one of the concurrent connections is terminated.

[0004] In accordance with a further aspect of the invention, a method is provided for controlled loading of a cluster-based server having a dispatcher and a plurality of back-end servers. The method includes defining, for each back-end server, a maximum number of concurrent connections that can

be supported, monitoring the performance of each back-end server, and dynamically adjusting the maximum number for at least one of the back-end servers in response to the monitoring to thereby adjust the performance of the cluster-based server.

In accordance with yet another aspect of the present invention, a computer server for providing data to clients includes a dispatcher for receiving data requests from a plurality of clients, and at least one back-end server. The dispatcher establishes at least one persistent connection with the back-end server, and forwards the data requests received from the plurality of clients to the back-end server over the persistent connection.

[0001] In accordance with another aspect of the present invention, a method for reducing connection overhead between a dispatcher and a server includes establishing a persistent connection between the dispatcher and the server, receiving at the dispatcher at least a first data request from a first client and a second data request from a second client, and forwarding the first data request and the second data request from the dispatcher to the server over the persistent connection.

[0002] In accordance with a further aspect of the present invention, a method for reducing connection overhead between a dispatcher and a server includes establishing a set of persistent connections between the dispatcher and the server, maintaining the set of persistent connections between the dispatcher and the server while establishing and terminating connections between the dispatcher and a plurality of clients, receiving at the dispatcher data requests from the plurality of clients over the connections between the dispatcher and the plurality of clients, and forwarding the received data requests from the dispatcher to the server over the set of persistent connections.

[0003] In accordance with a further aspect of the invention, a method for reducing back-end connection overhead in a cluster-based server includes establishing a set of persistent connections between a dispatcher and each of a plurality of back-end servers, maintaining each set of persistent connections while establishing and terminating connections between the dispatcher and a plurality of clients, receiving at the dispatcher data requests from the plurality of clients over the connections between the dispatcher and the plurality of clients, and forwarding each received data request from the dispatcher to one of the servers over one of the persistent connections.

[0015] In accordance with still another aspect of the present invention, a computer-readable medium has computer-executable instructions stored thereon for implementing any one or more of the servers and methods described herein.

[0016] Other aspects and features of the present invention will be in part apparent and in part pointed out hereinafter.

Brief Description of the Drawings

[0016] Fig. 1 is a block diagram of a server having an L7/3 dispatcher according to one embodiment of the present invention.

[0017] Fig. 2 is a block diagram of a cluster-based server having an L7/3 dispatcher according to another embodiment of the present invention.

[0018] Fig. 3 is a block diagram of a server having an L4/3 dispatcher according to a further embodiment of the present invention.

[0019] Fig. 4 is a block diagram of a cluster-based server having an L4/3 dispatcher according to yet another embodiment of the present invention.

[0020] Fig. 5 is a block diagram of a simplified TCP/IP protocol implemented by the L7/3 dispatcher of Fig. 2.

[0021] Fig. 6 is an activity diagram illustrating the processing of packets using the simplified TCP/IP protocol of Fig. 5.

[0022] Fig. 7(a) is a state diagram for the L7/3 dispatcher of Fig. 2 as it manages front-end connections.

[0023] Fig. 7(b) is a state diagram for the L7/3 dispatcher of Fig. 2 as it manages back-end connections.

[0024] Fig. 8 illustrates a two-dimensional server mapping array for storing connection information.

[0025] Fig. 9 is a block diagram illustrating the manner in which back-end connections are maintained.

[0026] Fig. 10 illustrates the manner in which the dispatcher of Fig. 2 translates sequence information for a packet passed from a back-end connection to a front-end connection.

[0027] Corresponding reference characters indicate corresponding features throughout the several views of the drawings.

Detailed Description of Preferred Embodiments:

[0028] A Web server according to one preferred embodiment of the present invention is illustrated in Fig. 1 and indicated generally by reference character 100. As shown in Fig. 1, the server 100 includes a dispatcher 102 and a back-end server 104 (the phrase "back-end server" does not require server 100 to be a cluster-based server). In this particular embodiment, the dispatcher 102 is configured to support open systems integration (OSI) layer seven (L7) switching (also known as content-based routing), and includes a queue 106 for storing data requests (e.g., HTTP requests) received from exemplary clients 108, 110, as further explained below. Preferably, the dispatcher 102 is transparent to both the clients 108, 110 and the back-end

server 104. That is, the clients perceive the dispatcher as a server, and the back-end server perceives the dispatcher as one or more clients.

[0029] The dispatcher 102 preferably maintains a front-end connection 112, 114 with each client 108, 110, and a dynamic set of persistent back-end connections 116, 118, 120 with the back-end server 104. The back-end connections 116-120 are persistent in the sense that the dispatcher 102 can forward multiple data requests to the back-end server 104 over the same connection. Also, the dispatcher can preferably forward data requests received from different clients to the back-end server 104 over the same connection, when desirable. This is in contrast to using client-specific back-end connections, as is done for example in prior art L7/3 cluster-based servers. As a result, back-end connection overhead is markedly reduced. Alternatively, non-persistent and/or client-specific back-end connections may be employed. The set of back-end connections 116-120 is dynamic in the sense that the number of connections maintained between the dispatcher 102 and the back-end server 104 may change over time, including while the server 100 is in use.

[0030] The front-end connections 112, 114 may be established using HTTP/1.0, HTTP/1.1 or any other suitable protocol, and may or may not be persistent.

[0031] Each back-end connection 116-120 preferably remains open until terminated by the back-end server 104 when no data request is received over that connection within a certain amount of time (e.g., as defined by HTTP/1.1), or until terminated by the dispatcher 102 as necessary to adjust the performance of the back-end server 104, as further explained below.

[0032] The back-end connections 116-120 are initially established using the HTTP/1.1 protocol (or any other

protocol supporting persistent connections) either before or after the front-end connections 112-114 are established. For example, the dispatcher may initially define and establish a default number of persistent connections to the back-end server before, and in anticipation of, establishing the front-end connections. This default number is typically less than the maximum number of connections that can be supported concurrently by the back-end server 104 (e.g., if the back-end server can support up to 256 concurrent connections, the default number may be five, ten, one hundred, etc., depending on the application). Preferably, this default number represents the number of connections that the back-end server 104 can readily support while yielding good performance. It should therefore be apparent that the default number of permissible connections selected for any given back-end server will depend upon that server's hardware and/or software configuration, and may also depend upon the particular performance metric (e.g., request rate, average response time, maximum response time, throughput, etc.) to be controlled, as discussed further below. Alternatively, the dispatcher 102 may establish the back-end connections on an as-needed basis (i.e., as data requests are received from clients) until the default (or subsequently adjusted) number of permissible connections for the back-end server 104 is established. When a back-end connection is terminated by the back-end server, the dispatcher may establish another back-end connection immediately, or when needed.

[0033] According to the present invention, the performance of a server may be enhanced by limiting the amount of data processed by that server at any given time. For example, by limiting the number of data requests processed concurrently by a server, it is possible to reduce the average response time and increase server throughput. Thus, in the

embodiment under discussion, the dispatcher 102 is configured to establish connections with clients and accept data requests therefrom to the fullest extent possible while, at the same time, limit the number of data requests processed by the back-end server 104 concurrently. In the event that the dispatcher 102 receives a greater number of data requests than what the back-end server 104 can process efficiently (as determined with reference to a performance metric for the back-end server), the excess data requests are preferably stored in the queue 106.

[0034] Once a data request is forwarded by the dispatcher 102 over a particular back-end connection, the dispatcher will preferably not forward another data request over that same connection until it receives a response to the previously forwarded data request. In this manner, the maximum number of data requests processed by the back-end server 104 at any given time can be controlled by dynamically controlling the number of back-end connections 116-120. Limiting the number of concurrently processed data requests prevents thrashing of server resources by the back-end server's operating system, which could otherwise degrade performance.

[0035] A back-end connection over which a data request has been forwarded, and for which a response is pending, may be referred to as an "active connection." A back-end connection over which no data request has as yet been forwarded, or over which no response is pending, may be referred to as an "idle connection."

[0036] Data requests arriving from clients at the dispatcher 102 are forwarded to the back-end server 104 for processing as soon as possible and, in this embodiment, in the same order that such data requests arrived at the dispatcher. Upon receiving a data request from a client, the dispatcher 102 selects an idle connection for forwarding that data request to the back-end server 104. When no idle connection

is available, data requests received from clients are stored in the queue 106. Thereafter, each time an idle connection is detected, a data request is retrieved from the queue 106, preferably on a FIFO basis, and forwarded over the formerly idle (now active) connection. Alternatively, the system may be configured such that all data requests are first queued, and then dequeued as soon as possible (which may be immediately) for forwarding to the back-end server 104 over an idle connection. After receiving a response to a data request from the back-end server 104, the dispatcher 102 forwards the response to the corresponding client.

[0037] Client connections are preferably processed by the dispatcher 102 on a first come, first served (FCFS) basis. When the number of data requests stored in the queue 106 exceeds a defined threshold, the dispatcher preferably denies additional connection requests (e.g., TCP requests) received from clients (e.g., by sending an RST to each such client). In this manner, the dispatcher 102 ensures that already established front-end connections 108-110 are serviced before requests for new front-end connections are accepted. When the number of data requests stored in the queue 106 is below a defined threshold, the dispatcher may establish additional front-end connections upon request until the maximum number of front-end connections that can be supported by the dispatcher 102 is reached, or until the number of data requests stored in the queue 106 exceeds the defined threshold.

[0038] As noted above, the dispatcher 102 maintains a variable number of persistent connections 116-120 with the back-end server 104. In essence, the dispatcher 102 implements a feedback control system by monitoring a performance metric for the back-end server 104 and then adjusting the number of back-end connections 116-120 as necessary to adjust the performance metric as desired. For example, suppose a

primary performance metric of concern for the back-end server 104 is overall throughput. If the monitored throughput falls below a minimum level, the dispatcher 102 may adjust the number of back-end connections 116-120 until the throughput returns to an acceptable level. Whether the number of back-end connections should be increased or decreased to increase server throughput will depend upon the specific configuration and operating conditions of the back-end server 104 in a given application. This decision may also be based on past performance data for the back-end server 104. The dispatcher 102 may also be configured to adjust the number of back-end connections 116-120 so as to control a performance metric for the back-end server 104 other than throughput, such as, for example, average response time, maximum response time, etc. For purposes of stability, the dispatcher 102 is preferably configured to maintain the performance metric of interest within an acceptable range of values, rather than at a single specific value.

[0039] In the embodiment under discussion, where all communications with clients 108-110 pass through the dispatcher 102, the dispatcher can independently monitor the performance metric of concern for the back-end server 104. Alternatively, the back-end server may be configured to monitor its performance and provide performance information to the dispatcher.

[0040] As should be apparent from the description above, the dispatcher 102 may immediately increase the number of back-end connections 116-120 as desired (until the maximum number of connections which the back-end server is capable of supporting is reached). To decrease the number of back-end connections, the dispatcher 102 preferably waits until a connection becomes idle before terminating that connection

(in contrast to terminating an active connection over which a response to a data request is pending).

[0041] The dispatcher 102 and the back-end server 104 may be implemented as separate components, as illustrated generally in Fig. 1. Alternatively, they may be integrated in a single computer device having at least one processor. For example, the dispatcher functionality may be integrated into a conventional Web server (having sufficient resources) for the purpose of enhancing server performance. In one particular implementation of this embodiment, the server 100 achieved nearly three times the performance, measured in terms of HTTP request rate, of a conventional Web server.

[0042] A cluster-based server 200 according to another preferred embodiment of the present invention is shown in Fig. 2, and is preferably implemented in manner similar to the embodiment described above with reference to Fig. 1, except as noted below. As shown in Fig. 2, the cluster-based server 200 employs multiple back-end servers 202, 204 for processing data requests provided by exemplary clients 206, 208 through an L7 dispatcher 210 having a queue 212. The dispatcher 210 preferably manages a dynamic set of persistent back end connections 214-218, 220-224 with each back-end server 202, 204, respectively. The dispatcher 210 also controls the number of data requests processed concurrently by each back-end server at any given time in such a manner as to improve the performance of each back-end server and, thus, the cluster-based server 200.

[0043] As in the embodiment of Fig. 1, the dispatcher 210 preferably refrains from forwarding a data request to one of the back-end servers 202-204 over a particular connection until the dispatcher 210 receives a response to a prior data request forwarded over the same particular connection (if applicable). As a result, the dispatcher 210 can control the maximum number of data requests processed by any back-

end server at any given time simply by dynamically controlling the number of back-end connections 214-224.

[0044] While only two back-end servers 202, 204 and two exemplary clients 206, 208 are shown in Fig. 2, those skilled in the art will recognize that additional back-end servers may be employed, and additional clients supported, without departing from the scope of the invention. Likewise, although Fig. 2 illustrates the dispatcher 210 as having three persistent connections 214-218, 220-224 with each back-end server 202, 204, it should be apparent from the description below that the set of persistent connections between the dispatcher and each back-end server may include more or less than three connections at any given time, and the number of persistent connections in any given set may differ at any time from that of another set.

[0045] The default number of permissible connections initially selected for any given back-end server will depend upon that server's hardware and/or software configuration, and may also depend upon the particular performance metric (e.g., request rate, throughput, average response time, maximum response time, etc.) to be controlled for that back-end server. Preferably, the same performance metric is controlled for each back-end server.

[0046] An "idle server" refers to a back-end server having one or more idle connections, or to which an additional connection can be established by the dispatcher without exceeding the default (or subsequently adjusted) number of permissible connections for that back-end server.

[0047] Upon receiving a data request from a client, the dispatcher preferably selects an idle server, if available, and then forwards the data request to the selected server. If no idle server is available, the data request is stored in the queue 212. Thereafter, each time an idle connection is detected, a data request is retrieved from the queue 212,

preferably on a FIFO basis, and forwarded over the formerly idle (now active) connection. Alternatively, the system may be configured such that all data requests are first queued and then dequeued as soon as possible (which may be immediately) for forwarding to an idle server.

[0048] To the extent that multiple idle servers exist at any given time, the dispatcher preferably forwards data requests to these idle servers on a round-robin basis. Alternatively, the dispatcher can forward data requests to the idle servers according to another load sharing algorithm, or according to the content of such data requests (i.e., content-based dispatching). Upon receiving a response from a back-end server to which a data request was dispatched, the dispatcher forwards the response to the corresponding client.

[0049] A Web server according to another preferred embodiment of the present invention is illustrated in Fig. 3 and indicated generally by reference character 300. Similar to the server 100 of Fig. 1, the server 300 of Fig. 3 includes a dispatcher 302 and a back-end server 304. However, in this particular embodiment, the dispatcher 302 is configured to support open systems integration (OSI) layer four (L4) switching. Thus, connections 314-318 are made between exemplary clients 308-312 and the back-end server 304 directly rather than with the dispatcher 302. The dispatcher 302 includes a queue 306 for storing connection requests (e.g., SYN packets) received from clients 308-312.

[0050] Similar to other preferred embodiments described above, the dispatcher 302 monitors a performance metric for the back-end server 304 and controls the number of connections 314-318 established between the back-end server 304 and clients 308-312 to thereby control the back-end server's performance. Preferably, the dispatcher 302 is an L4/3 dispatcher (i.e., it implements layer 4 switching with layer

3 packet forwarding), thereby requiring all transmissions between the back-end server 304 and clients 308-312 to pass through the dispatcher. As a result, the dispatcher 302 can monitor the back-end server's performance directly. Alternatively, the dispatcher can monitor the back-end server's performance via performance data provided to the dispatcher by the back-end server, or otherwise.

[0051] The dispatcher 302 monitors a performance metric for the back-end server 304 (e.g., average response time, maximum response time, server packet throughput, etc.) and then dynamically adjusts the number of connections 314-318 to the back-end server 304 as necessary to adjust the performance metric as desired. The number of connections is dynamically adjusted by controlling the number of connection requests (e.g., SYN packets), received by the dispatcher 302 from clients 308-312, that are forwarded to the back-end server 304.

[0052] Once a default number of connections 314-318 are established between the back-end server 304 and clients 308-312, additional connection requests received at the dispatcher 302 are preferably stored in the queue 306 until one of the existing connections 314-318 is terminated. At that time, a stored connection request can be retrieved from the queue 306, preferably on a FIFO basis, and forwarded to the back-end server 304 (assuming the dispatcher has not reduced the number of permissible connections to the back-end server). The back-end server 304 will then establish a connection with the corresponding client and process data requests received over that connection.

[0053] Fig. 4 illustrates a cluster-based embodiment of the Web server 300 shown in Fig. 3. As shown in Fig. 4, a cluster-based server 400 includes an L4/3 dispatcher 402 having a queue 404 for storing connection requests, and several back-end servers 406, 408. As in the embodiment of

Fig. 3, connections 410-420 are made between exemplary clients 422, 424 and the back-end servers 406, 408 directly. The dispatcher 402 preferably monitors the performance of each back-end server 406, 408 and dynamically adjusts the number of connections therewith, by controlling the number of connection requests forwarded to each back-end server, to thereby control their performance.

[0054] A detailed implementation of the L7/3 cluster-based server 200 shown in Fig. 2 will now be described with reference to Figs. 5-10. All functions of the dispatcher 210 are preferably implemented via a software application implementing a simplified TCP/IP protocol, shown in Fig. 5, and running in user-space (in contrast to kernel space) on commercially off-the-shelf ("COTS") hardware and operating system software. In one preferred embodiment, this software application runs under the Linux operating system or another modern UNIX system supporting *libpcap*, a publicly available packet capture library, and POSIX threads. As a result, the dispatcher can capture the necessary packets in the datalink layer.

[0055] When a packet arrives at the datalink layer of the dispatcher 210, the packet is preferably applied to each filter defined by the dispatcher, as shown in Figure 5. The packet capture device then captures all the packets in which it is interested. For example, the packet capture device can operate in a promiscuous mode, during which all packets arriving at the datalink layer are copied to a packet capture buffer and then filtered, through software, according to, e.g., their source IP or MAC address, protocol type, etc. Matching packets can then be forwarded to the application making the packet capture call, whereas non-matching packets can be discarded. Alternatively, packets arriving at the datalink layer can be filtered through hardware (e.g., via a network interface card) in addition to

or instead of software filtering. In the latter case, interrupts are preferably generated at the hardware level only when broadcast packets or packets addressed to that hardware are received.

[0056] In this embodiment, two packet capture devices are used to capture packets from the clients 206-208 and the back-end servers 202-204, respectively. These packets are then decomposed and analyzed using the simplified TCP/IP protocol, as further described below. Packets seeking to establish or terminate a connection are preferably handled by the dispatcher 210 immediately. Packets containing data requests (e.g., HTTP requests) are stored in the queue 212 when all of the back-end connections 214-224 are active. When an idle server is detected, a data request is dequeued, combined with corresponding TCP and IP headers, and sent to this server using a raw socket (raw socket is provided in many operating systems, e.g., UNIX, for users to read and write raw network protocol datagrams with a protocol field that is not processed by the kernel). Packets containing response data from a back-end server are combined with appropriate TCP and IP headers and passed to the corresponding client using raw sockets. This process is illustrated by the activity diagram of Figure 6.

[0057] The simplified TCP/IP protocol implemented in the dispatcher application software will now be described. The primary use of the IP protocol is to obtain the source and destination addresses of packets. Because, in this particular embodiment, the dispatcher and the back-end servers are interconnected through a local area network (LAN), the maximum transmission unit (MTU) of the TCP segment is small and does not require fragmentation when it arrives at the IP layer. Therefore, IP refragmentation is omitted. Additionally, due to the properties of the front-

end connections and the back-end connections, the following TCP specifications are simplified or omitted:

a. Sequence Number Space. Sequence space is used for sequencing the data transmitted. In UNIX, about thirteen variables are used to implement the sequence window scaling and sliding. All packets transmitted to establish and terminate a connection are short and in sequence except for retransmitted packets. Once a request has been assigned to a server, which is when bulk data transmission occurs, the dispatcher acts like a gateway, whose function is too simply change packet header fields and pass packets. Thus, the sequence window in this embodiment is simplified to have a size of one, to deal with connection setup and termination.

b. Timers.

1. *Retransmission.* Retransmission is done in TCP to avoid data loss when the sender does not receive an acknowledgement within a certain period. Since the back-end servers are distributed in the same LAN, data loss is rare. When establishing a connection with a client, since the client is active, the client will retransmit the same packet if it does not receive the packet from the dispatcher. When terminating a connection with the client, if the dispatcher does not receive any response from the client for a certain period, the dispatcher will disconnect the connection. Therefore, retransmission can be omitted.

2. *Persist timer.* This is set when the other end of a connection advertises a window of zero, thereby stopping TCP from sending data. When it expires, one byte of data is sent to determine if the window has opened. This is not applicable since

the bulk data transmission will not occur when establishing and terminating connections.

3. *Delayed acknowledgement*. This is used to improve the efficiency of the transmission. It is not applicable to establishing and terminating connections because an immediate response can be given, but could be used to acknowledge an HTTP request. Because maintaining an alarm or maintaining a time record and polling for each connection is expensive, this problem is solved by sending an acknowledgement to each HTTP request immediately after it is received.

c. Option Field. Three options are implemented in UNIX TCP: MSS (Maximum Segment Size), window scale, and timestamp. For simplicity, only the MSS option is implemented in this embodiment.

d. State Diagram. General TCP implementations consider all possible applications a host may have. For a Web server, some transitions may not happen at all. In this Web embodiment, the following scenarios are assumed not to happen: simultaneous open for front-end connections and for back-end connections; and simultaneous close for back-end connections. CLOSE_WAIT is also not implemented, as an immediate response can be sent to acknowledge the FIN flag without waiting for the application to finish its work before sending the FIN flag. State diagrams for the dispatcher 210 as it manages front-end and back-end connections are shown in Figs. 7(a) and 7(b), respectively.

[0058] The preferred manner in which the dynamic sets of persistent back-end connections are managed will now be described with reference to Figs. 8 and 9. As shown in Fig. 8, a two-dimensional server-mapping array is used to store the connection information between the dispatcher and the

back-end servers. Alternatively, a linked list could be used. Each server is preferably associated with a unique index number, and newly added servers are assigned larger index numbers. Each connection to a back-end server is identified by a port number, which is used by the dispatcher to set up that connection.

[0059] A third dimension, port number layer, is preferably used to keep the number of connections fixed. For example, when a client connects to an Apache server using HTTP/1.1, the server will close the connection when it receives a bad request, which may be a non-existent URL. In this situation, the connection becomes unusable for a certain period of time (which varies by operating system). This means the port number is disabled. In order to maintain the active connection number, a new connection to the same server is preferably opened. Thus, a new memory space must be allocated for the connection. To efficiently use memory space and manage the connection set, the port number manager uses layers to assign a different port number and stores its information in the same slot. As shown in Figure 8, a port number is uniquely determined by the index of the server, the connection index of this server, the index of port number layers, and the port start number. According to this approach, if the port start number is defined as 10000, then the port number used by the dispatcher to setup the first connection to the first back-end server will be 10000 and the second connection to the first back-end server will be 10001. If the number of permissible connections to a particular back-end server is, for example, eight, then the port number used by the dispatcher to setup the first connection to the second back-end server is 10008. If the maximum port layer number is five and the maximum server number is 256, then the maximum port number used to connect to a back-end server will be 10000 + 5 * 8 * 256 - 1 =

10239. The port number used by the dispatcher to setup any given connection can be determined from the following equation: dispatcher port number (dport) = iLayer * nServer * nServerConn + iServer * nServerConn +iServerConn, where i ranges from 0 to n - 1, and n represents the number of layers, servers, and maximum number of connections per server, respectively. In other words, there are three different "n" values: one for the maximum number of layers; one for the maximum number of servers; and one for the maximum number of connections allowed per server.

[0060] To maintain the dynamic sets of connections with the back-end servers efficiently, two queues are preferably used: a not-in-use queue 902; and an idle queue 904. In this particular implementation in which back-end connections are established on an as-needed basis (rather than, e.g., initially establishing a default number of connections), all port numbers are initially inserted into the not-in-use queue 902 in such a way that each back-end server has an equal chance to be connected to by the dispatcher. When the dispatcher receives a connection request from a client, it removes a port number from the head of the not-in-use queue 902 and uses it to set up a connection with the corresponding back-end server. This port number is placed in the idle queue 904 once the connection is established. When a data request arrives from a client, the dispatcher matches the data request with an idle port, dequeues the associated port number from the idle queue 904, and forwards the data request to the back-end server associated with the dequeued port number. When the load of the dispatcher decreases and one or more back-end connections do not receive a data request within a certain time interval (which is three minutes in this particular implementation), this back-end connection(s) is terminated by the corresponding back-end server(s), and the corresponding port numbers are

placed back into the not-in-use queue 902. Thus, the idle queue stores port numbers associated with idle connections, and the not-in-use queue stores port numbers not associated with an existing connection. In this manner, the network resources and the resources of the back-end servers are used efficiently.

[0061] The preferred manner in which connections are made between the dispatcher and clients will now be described. Information associated with these connections is preferably maintained using a hash table. Each hash entry is uniquely identified by a tuple of client IP address, client port number, and a dispatcher port number. To calculate the hash value, the client IP address and the client port number are used to get a hash index. Collision is handled using open addressing, which resolves the collision problems by polling adjacent slots until an empty one is found. To obtain the hash entry, the client IP address and port number are compared to those of entries in the hash slot. The dispatcher port numbers preferably have a one-to-one relationship with back-end servers. The hash index or map index that stores the information for a particular connection is preferably stored in the data request queue 212 shown in Fig. 2. Each time a hash index is dequeued, the corresponding connection is found, and the head of its request list is dispatched to a back-end server. This index is stored in the server-mapping table for mapping the response to the connection. After the response from a back-end server is acknowledged, the data request is discarded and the connection is either terminated (for HTTP/1.0 sessions) or placed in the data request queue 212.

[0062] According to the TCP protocol specification, a sequence number space is maintained by each side of a connection to control the transmission. When a packet arrives from a back-end server, it includes sequence information specific

to the connection between the back-end server and the dispatcher. This packet must then be changed by the dispatcher to carry sequence information specific to the front-end connection between the dispatcher and the associated client. Fig. 10 provides an example of how the packet sequence number is changed while it is passed by the dispatcher. The four sequence numbers are represented using the following symbols:

X–the sequence number of the next byte to be sent to the client by the dispatcher.

Y–the sequence number of the next byte to be sent to the dispatcher by the client.

A–the sequence number of the next byte to be sent to the server by the dispatcher.

B–the sequence number of the next byte to be sent to the dispatcher by the server.

In step (1), after the dispatcher sends a client's request to a selected back-end server, it saves the initial sequence numbers X0 and B0. In step (2), the dispatcher receives the acknowledgement from the selected server so it increases A0 to A1 (A1 = A0 + n1, where n1 is the request size, or number of bytes, sent to the back-end server). In step (3), the dispatcher receives the first response packet from the back-end server with the sequence number B0 and the acknowledgement number A1. Since this is the first response, the dispatcher searches the header of the packet for content-length field and records the total bytes that the server is sending to the client. In step (4), the dispatcher changes the sequence number to X0 and the acknowledgement number to Y0 and forwards the packet to the client. The address space and checksum of the packet are also updated accordingly every time the packet is passed. In step (5), the dispatcher receives the acknowledgement from the client with the sequence number Y0 and the

acknowledgement number Z. The dispatcher compares Z with X0; if Z>X0, then the dispatcher updates X0 to X1; otherwise, it keeps X0. In step (6), the dispatcher changes the sequence number to A1 and the acknowledgment number to B1 and sends it to the back-end server. B1 is determined by B0 and the difference between X1 and X0, which represents the number of bytes that the client has received. Thus, B1 = B0 + X1 - X0. Based on this acknowledgement, the dispatcher calculates the remaining packet length to be received. Since the remaining packet length is greater than zero, the dispatcher waits for the next packet. In step (7), the dispatcher receives the second response packet from the server with the sequence number B1 (assuming the length of the first packet is n2, then B1 = B0 + n2) and the acknowledgment number A1. In step (8), the dispatcher changes the sequence number to X1 and the acknowledgement number to Y0 and sends the packet to the client. In step (9), the dispatcher receives the acknowledgment from the client and repeats the same work done in step (5). In step (10), the dispatcher repeats the functions performed in step (6).

[0063] From the foregoing description, it should be understood that the dispatcher preferably does not acknowledge the amount of data it receives from the server. Instead, it passes the packet on to the client and acknowledges it only after it receives the acknowledgement from the client. In this way, the server is responsible for the retransmission when it has not received an acknowledgment within a certain period, and the client is responsible for the flow control if it runs out of buffer space.

[0064] According to the TCP protocol specification, the TIME_WAIT state is provided for a sender to wait for a period of time to allow the acknowledgement packet sent by the sender to die out in the network. A soft timer and a

queue are preferably used to keep track of this time interval. When a connection enters the TIME_WAIT state, its hash index is placed in the TIME_WAIT queue. The queue is preferably checked every second if the interval exceeds a certain period. For UNIX, this interval is one minute, but in the particular implementation of the invention under discussion, because of the short transmission time and short route, it is preferably set to one second. The soft timer, which is realized by reading the system time each time after the program has finished processing one packet, is preferably used instead of a kernel alarm to eliminate the overhead involved in the interrupt caused by the kernel.

[0065] While the present invention has been described primarily in a Web server context, those skilled in the art will recognize that the teachings of the invention are applicable to other server applications as well.

[0066] When introducing elements of the present invention or the preferred embodiment(s) thereof, the articles "a", "an", "the" and "said" are intended to mean that there are one or more such elements. The terms "comprising", "including" and "having" are intended to be inclusive and mean that there may be additional elements other than those listed.

[0067] As various changes could be made in the above constructions without departing from the scope of the invention, it is intended that all matter contained in the above description or shown in the accompanying drawings shall be interpreted as illustrative and not in a limiting sense.

What is claimed is:

1. A server for providing data to clients, the server comprising:

a dispatcher having a queue for storing requests received from clients; and

at least one back-end server;

wherein the dispatcher stores in the queue one or more of the requests received from clients when the back-end server is unavailable to process said one or more requests;

wherein the dispatcher retrieves said one or more requests from the queue for forwarding to the back-end server when the back-end server becomes available to process said one or more requests; and

wherein the dispatcher determines whether the back-end server is available to process said one or more requests by comparing a number of connections concurrently supported by the back-end server to a maximum number of concurrent connections that the back-end server is permitted to support, the maximum number being less than a maximum number of connections which the back-end server is capable of supporting concurrently.

2. The server of claim 2 wherein the dispatcher is configured to monitor a performance of the back-end server, to define the maximum number of concurrent connections that the back-end server is permitted to support, and to dynamically adjust the maximum number in response to the monitored performance.

3. The server of claim 1 wherein the server is a cluster-based server comprising a plurality of back-end servers, the dispatcher is configured to store in the queue said one or more requests when none of the back-end servers are available to process said one or more requests, and the

dispatcher is further configured to retrieve said one or more requests from the queue for forwarding to one of the back-end servers when said one of the back-end servers becomes available to process said one or more requests.

4. The server of claim 1 wherein the server is a Web server.

5. The server of claim 1 wherein the dispatcher and the back-end server are implementing using COTS hardware.

6. The server of claim 1 wherein the dispatcher comprises a first computer device, the back-end server comprises a second computer device, and the first and second computer devices are configured to communicate with one another over a computer network.

7. The server of claim 1 wherein the dispatcher is an OSI layer 7 dispatcher and said requests are data requests.

8. The server of claim 7 wherein the dispatcher implements a simplified TCP/IP protocol in user-space.

9. The server of claim 1 wherein the dispatcher is an OSI layer 4 dispatcher and said requests are connection requests.

10. A computer-readable medium having computer-executable instructions for performing the method of claim 1.

11. A method for controlled server loading, the method comprising the steps of:

defining a maximum number of concurrent connections that a server is permitted to support;

limiting a number of concurrent connections supported by the server to the maximum number;

monitoring the server's performance while it supports the concurrent connections; and

dynamically adjusting the maximum number as a function of the server's performance to thereby control a performance factor for the server.

12. The method of claim 11 wherein the defining step includes defining the maximum number to be less than a maximum number of connections which the server is capable of supporting concurrently.

13. The method of claim 11 wherein the concurrent connections are connections between the server and clients.

14. The method of claim 11 wherein the concurrent connections are connections between the server and a dispatcher.

15. The method of claim 11 wherein the server is a back-end server in a cluster-based server having a dispatcher, and the dynamically adjusting step includes dynamically adjusting the maximum number of concurrent connections that can be established between the back-end server and the dispatcher.

16. The method of claim 15 wherein each concurrent connection is a persistent connection over which data requests from multiple clients can be sent by the dispatcher to the back-end server.

17. The method of claim 11 wherein the dynamically adjusting step includes dynamically adjusting the maximum number in response to the monitoring step such that the server operates at or above a minimum performance level.

18. The method of claim 17 wherein the monitoring step includes monitoring the server's performance level in terms of a performance metric selected from the group consisting of request rate, average response time, maximum response time and server throughput.

19. A method for controlled server loading, the method comprising the steps of:

receiving a plurality of data requests from clients;

forwarding a number of the data requests to a server for processing; and

storing at least one of the data requests until the server completes processing at least one of the forwarded data requests.

20. The method of claim 19 further comprising the steps of retrieving the stored data request after the server completes processing at least one of the forwarded data requests, and forwarding the retrieved data request to the server for processing.

21. The method of claim 19 wherein the storing step includes storing a plurality of the data requests, the method further comprising the step of retrieving one of the stored data requests and forwarding the retrieved one of the data requests to the server for processing each time the server completes processing one of the forwarded data requests.

22. The method of claim 21 wherein the retrieving step includes retrieving the stored data requests on a FIFO basis.

23. The method of claim 19 wherein the data requests are HTTP requests.

24. The method of claim 19 wherein the receiving, forwarding and storing steps are performed by a single computer device having at least one processor.

25. The method of claim 24 wherein the single computer device comprises the server.

26. The method of claim 19 wherein the storing step is performed by a dispatcher and includes storing at least one of the data requests until the dispatcher receives a response from the server to at least one of the forwarded data requests.

27. A method for controlled server loading, the method comprising the steps of:

defining a maximum number of data requests that a server is permitted to process concurrently;

monitoring the server's performance; and

dynamically adjusting the maximum number in response to the monitoring step to thereby adjust the server's performance.

28. The method of claim 27 wherein the monitoring step includes monitoring the server's performance in terms of a performance metric selected from the group consisting of request rate, average response time, maximum response time, and server throughput.

29. The method of claim 27 further comprising the steps of receiving a plurality of data requests from clients, forwarding some of the data requests to the server for processing, and storing at least one of the data requests until the server completes processing one of the forwarded data requests.

30. The method of claim 27 wherein the defining step includes defining a maximum number of connections that can be supported concurrently by the server and limiting the number of data requests that can be pending on each connection.

31. The method of claim 30 wherein the defining step includes limiting the number of data requests that can be pending on each connection to one.

32. A method for controlled loading of a cluster-based server, the cluster-based server including a dispatcher and a plurality of back-end servers, the method comprising the steps of:

receiving at the dispatcher a plurality of data requests from clients;

forwarding a plurality of the data requests to each of the back-end servers for processing; and

storing at the dispatcher at least one of the data requests until one of the back-end servers completes processing one of the forwarded data requests.

33. The method of claim 32 wherein the storing step includes storing a plurality of the data requests and the forwarding step includes forwarding one of the stored data requests to one of the back-end servers each time one of the

back-end servers completes processing one of the forwarded data requests.

34. The method of claim 32 wherein the cluster-based server is an L7/3 server.

35. A method for controlled loading of a cluster-based server, the cluster-based server including a dispatcher and a plurality of back-end servers, the method comprising the steps of:

defining, for each back-end server, a maximum number of data requests that can be processed concurrently;

monitoring the performance of each back-end server; and

dynamically adjusting the maximum number for at least one of the back-end servers in response to the monitoring step to thereby adjust the performance of the cluster-based server.

36. The method of claim 35 wherein the dynamically adjusting step includes dynamically adjusting the maximum number for each back-end server.

37. The method of claim 35 wherein the dynamically adjusting step includes dynamically adjusting the maximum number for said one of the back-end servers as a function of that back-end server's performance.

38. The method of claim 35 further comprising the steps of receiving a plurality of data requests from clients, forwarding some of the data requests to the back-end servers for processing, and storing at least one of the data requests until one of the back-end servers completes processing one of the forwarded data requests.

39. A server for providing data to clients, the server comprising:

an OSI layer 4 dispatcher having a queue for storing connection requests received from clients; and

at least one back-end server;

wherein the dispatcher stores in the queue one or more of the connection requests received from clients when the back-end server is unavailable to process said one or more connection requests;

wherein the dispatcher retrieves said one or more connection requests from the queue for forwarding to the back-end server when the back-end server becomes available to process said one or more connection requests; and

wherein the dispatcher determines whether the back-end server is available to process said one or more connection requests by comparing a number of connections concurrently supported by the back-end server to a maximum number of concurrent connections that the back-end server is permitted to support, the maximum number being less than a maximum number of connections which the back-end server is capable of supporting concurrently.

40. The server of claim 39 wherein the dispatcher is configured to monitor a performance of the back-end server, to define the maximum number of concurrent connections that the back-end server is permitted to support, and to dynamically adjust the maximum number in response to the monitored performance.

41. The server of claim 39 wherein the server is a cluster-based server comprising a plurality of back-end servers, wherein the dispatcher is configured to store in the queue said one or more connection requests when none of the back-end servers is available to process said one or

more connection requests, and wherein the dispatcher is further configured to retrieve said one or more connection requests from the queue for forwarding to one of the back-end servers when said one of the back-end servers becomes available to process said one or more connection requests.

42. The server of claim 39 wherein the server is a Web server.

43. The server of claim 39 wherein the dispatcher and the back-end server are embodied in COTS hardware.

44. The server of claim 39 wherein the dispatcher comprises a first computer device, wherein the back-end server comprises a second computer device, and wherein the first and second computer devices are configured to communicate with one another over a computer network.

45. A method for controlled server loading, the method comprising:

receiving a plurality of connection requests from clients;

establishing, in response to some of the connection requests, a number of concurrent connections between a server and clients; and

storing at least one of the connection requests until one of the established connections is terminated.

46. The method of claim 45 wherein the number of concurrent connections established between the server and clients is less than a maximum number of connections which the server is capable of supporting concurrently.

47. The method of claim 45 further comprising retrieving the stored connection request after at least one of the established connections is terminated, and establishing a connection between the server and a client associated with the retrieved connection request.

48. The method of claim 45 wherein the storing includes storing a plurality of the connection requests, the method further comprising retrieving one of the stored connection requests and establishing a new connection between the server and a client associated with the retrieved one of the connection requests each time one of the established connections is terminated.

49. The method of claim 48 wherein the retrieving includes retrieving the stored connection requests on a FIFO basis.

50. The method of claim 45 wherein the connection requests are TCP requests.

51. The method of claim 45 wherein at least the receiving and the storing are performed by a single computer device having at least one processor.

52. The method of claim 51 wherein the single computer device comprises the server.

53. A computer-readable medium having computer-executable instructions for performing the method of claim 45.

54. A method for controlled server loading, the method comprising:

defining a maximum number of concurrent connections that a server is permitted to support;

monitoring the server's performance; and

dynamically adjusting the maximum number in response to the monitoring to thereby adjust the server's performance.

55. The method of claim 54 wherein the monitoring includes monitoring the server's performance in terms of a performance metric selected from the group consisting of average response time, maximum response time, and server packet throughput.

56. The method of claim 54 further comprising receiving a plurality of connection requests from clients, establishing in response to some of the connection requests the maximum number of concurrent connections with the server, and storing at least one of the connection requests until one of the established connections is terminated.

57. The method of claim 56 wherein the dynamically adjusting includes dynamically adjusting the maximum number as a function of the number of connection requests that are concurrently stored.

58. The method of claim 57 wherein the dynamically adjusting includes increasing the maximum number when the number of concurrently stored connection requests is greater than a predefined number.

59. The method of claim 57 wherein the dynamically adjusting includes decreasing the maximum number when the number of concurrently stored connection requests is less than a predefined number.

60. A method for controlled loading of a cluster-based server, the cluster-based server including a dispatcher and a plurality of back-end servers, the method comprising:

receiving at the dispatcher a plurality of connection requests from clients;

forwarding a plurality of the connection requests to each of the back-end servers, each back-end server establishing a number of concurrent connections with clients in response to the connection requests forwarded thereto; and

storing at the dispatcher at least one of the connection requests until one of the concurrent connections is terminated.

61. The method of claim 60 wherein the storing includes storing a plurality of the connection requests, and wherein the forwarding includes forwarding one of the stored connection requests to one of the back-end servers each time one of the concurrent connections is terminated.

62. The method of claim 60 wherein the cluster-based server is an L4/3 server.

63. A method for controlled loading of a cluster-based server, the cluster-based server including a dispatcher and a plurality of back-end servers, the method comprising:

defining, for each back-end server, a maximum number of concurrent connections that can be supported;

monitoring the performance of each back-end server; and

dynamically adjusting the maximum number for at least one of the back-end servers in response to the monitoring to thereby adjust the performance of the cluster-based server.

64. The method of claim 63 wherein the dynamically adjusting includes dynamically adjusting the maximum number for each back-end server.

65. The method of claim 63 further comprising receiving a plurality of connection requests from clients, forwarding some of the connection requests to the back-end servers, each back-end server establishing a number of concurrent connections with clients in response to the connection requests forwarded thereto, and storing at least one of the connection requests until one of the concurrent connections is terminated.

66. The method of claim 65 wherein the dynamically adjusting includes dynamically adjusting the maximum number for said one of the back-end servers as a function of the number of connection requests that are concurrently stored.

67. A computer server for providing data to clients, the server comprising:

a dispatcher for receiving data requests from a plurality of clients; and

at least one back-end server;

wherein the dispatcher establishes at least one persistent connection with the back-end server, and forwards the data requests received from the plurality of clients to the back-end server over the persistent connection.

68. The computer server of claim 67 wherein the dispatcher includes a queue for storing the data requests until the back-end server is available for processing the data requests.

69. The computer server of claim 67 wherein the dispatcher is configured to establish a set of persistent connections with the back-end server.

70. The computer server of claim 69 wherein the dispatcher is configured to maintain the set of persistent connections with the back-end server while establishing and terminating connections between the dispatcher and the plurality of clients.

71. The computer server of claim 67 wherein the persistent connection is an HTTP/1.1 connection.

72. The computer server of claim 67 wherein the computer server is a Web server.

73. The computer server of claim 67 wherein the dispatcher and the back-end server are embodied in COTS hardware.

74. The computer server of claim 67 wherein the dispatcher comprises a first computer device, wherein the back-end server comprises a second computer device, and wherein the first and second computer devices are configured to communicate with one another over a computer network.

75. A method for reducing connection overhead between a dispatcher and a server, the method comprising:

establishing a persistent connection between the dispatcher and the server;

receiving at the dispatcher at least a first data request from a first client and a second data request from a second client; and

forwarding the first data request and the second data request from the dispatcher to the server over the persistent connection.

76. The method of claim 75 wherein the forwarding includes forwarding the second data request from the dispatcher to the server after the dispatcher receives from the server a response to the first data request.

77. The method of claim 76 further comprising storing the second data request at least until the dispatcher receives the response to the first data request.

78. The method of claim 76 wherein no data request is forwarded from the dispatcher to the server over the persistent connection between the first data request and the second data request.

79. The method of claim 75 wherein the persistent connection is an HTTP/1.1 connection.

80. A method for reducing connection overhead between a dispatcher and a server, the method comprising:

establishing a set of persistent connections between the dispatcher and the server;

maintaining the set of persistent connections between the dispatcher and the server while establishing and terminating connections between the dispatcher and a plurality of clients;

receiving at the dispatcher data requests from the plurality of clients over the connections between the dispatcher and the plurality of clients; and

forwarding the received data requests from the dispatcher to the server over the set of persistent connections.

81. The method of claim 80 wherein the dispatcher is an L7/3 dispatcher.

82. The method of claim 80 wherein the data requests are HTTP requests.

83. A method for reducing back-end connection overhead in a cluster-based server, the method comprising:

establishing a set of persistent connections between a dispatcher and each of a plurality of back-end servers;

maintaining each set of persistent connections while establishing and terminating connections between the dispatcher and a plurality of clients;

receiving at the dispatcher data requests from the plurality of clients over the connections between the dispatcher and the plurality of clients; and

forwarding each received data request from the dispatcher to one of the servers over one of the persistent connections.

84. The method of claim 83 wherein the dispatcher is an L7/3 dispatcher.

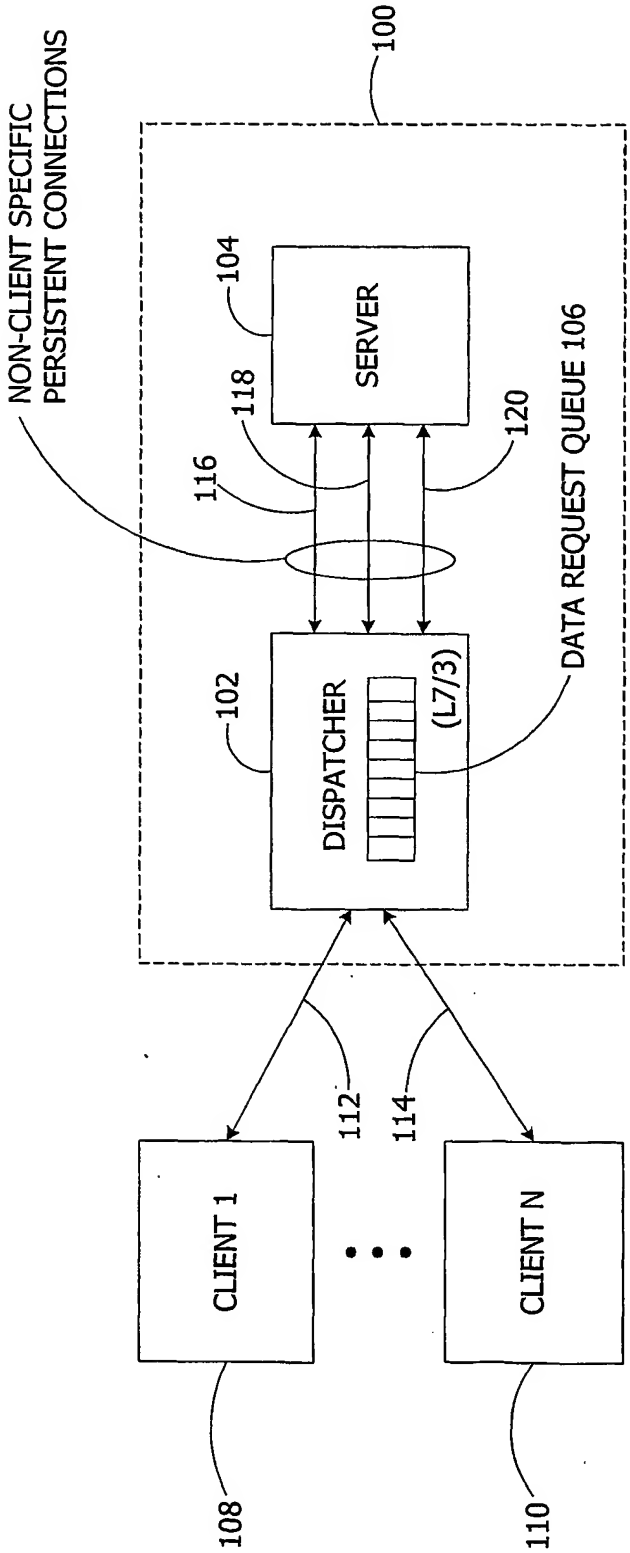

100
NON-CLIENT SPECIFIC
PERSISTENT CONNECTIONS
104
SERVER
118
120
116
DATA REQUEST QUEUE 106
102
DISPATCHER
(L7/3)
112
114
CLIENT 1
CLIENT N
108
110

FIG. 1

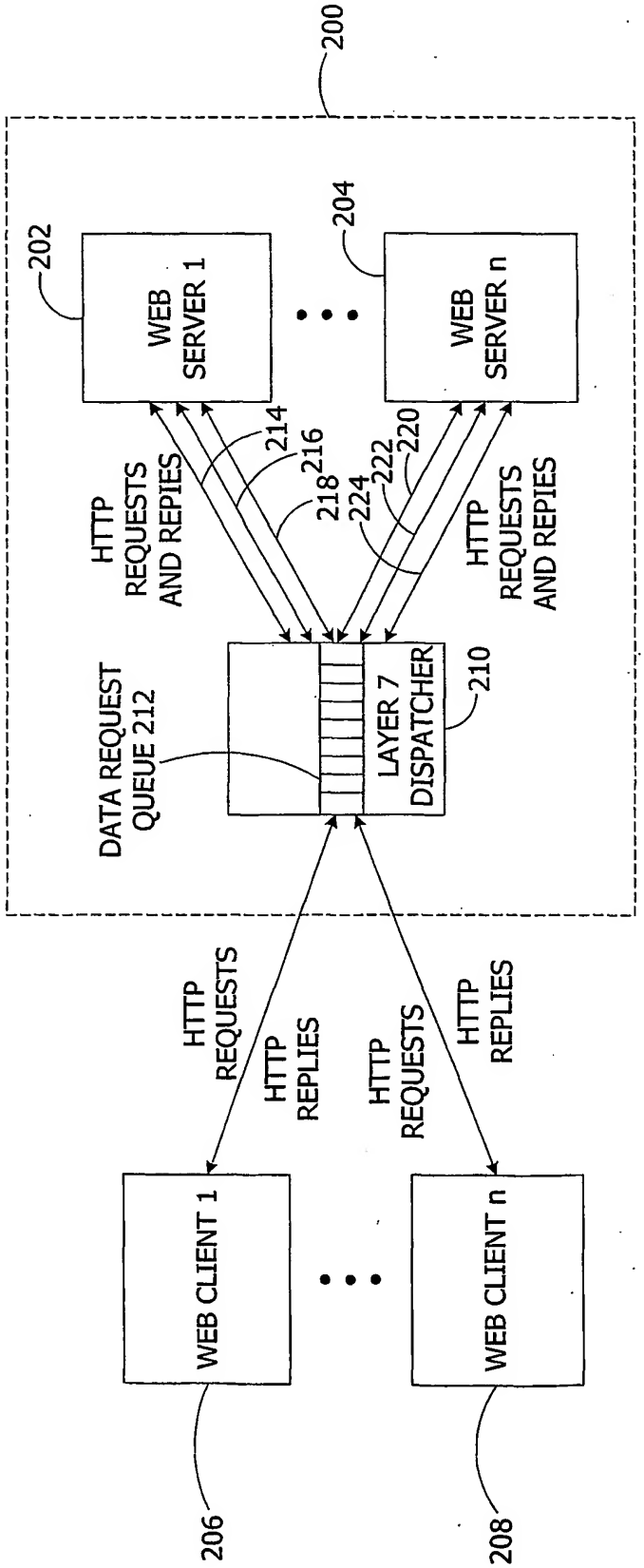

200
202
WEB SERVER 1
204
WEB SERVER n
HTTP REQUESTS AND REPIES
214
216
218
220
222
224
HTTP REQUESTS AND REPIES
DATA REQUEST QUEUE 212
LAYER 7 DISPATCHER
210
HTTP REQUESTS
HTTP REPLIES
HTTP REQUESTS
HTTP REPLIES
WEB CLIENT 1
WEB CLIENT n
206
208

FIG. 2

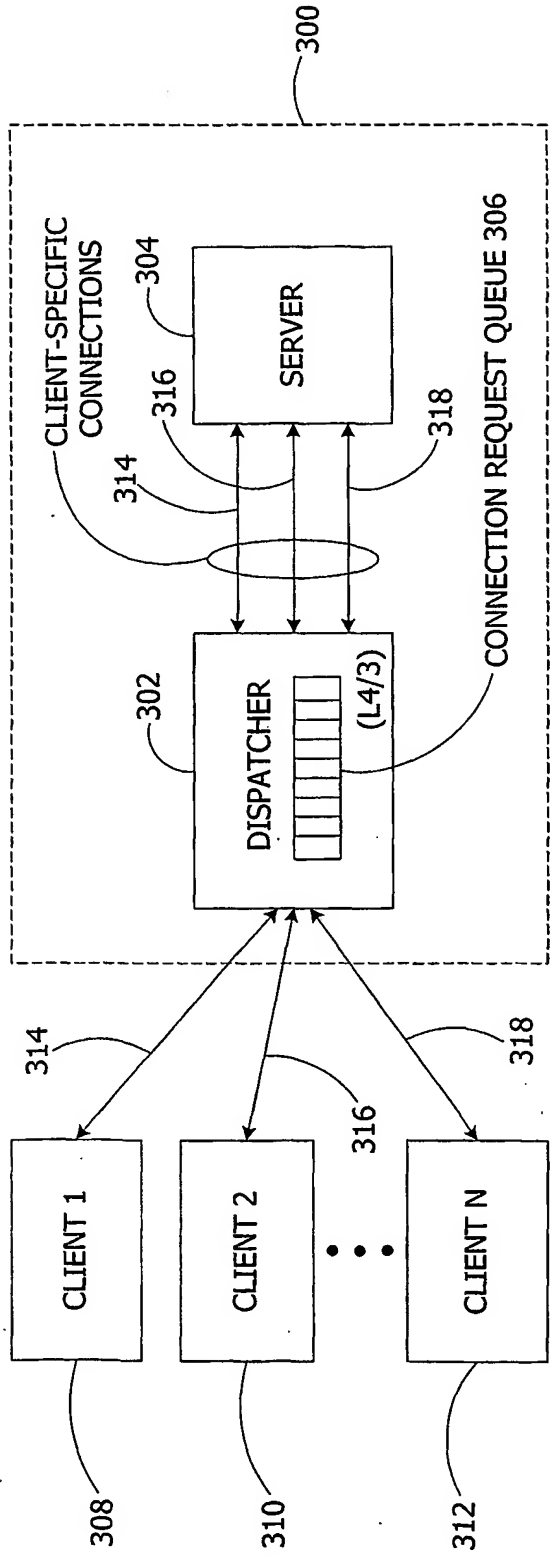

300
CLIENT-SPECIFIC CONNECTIONS
304
SERVER
316
318
314
CONNECTION REQUEST QUEUE 306
302
DISPATCHER
(L4/3)
314
316
318
CLIENT 1
CLIENT 2
CLIENT N
308
310
312

FIG. 3

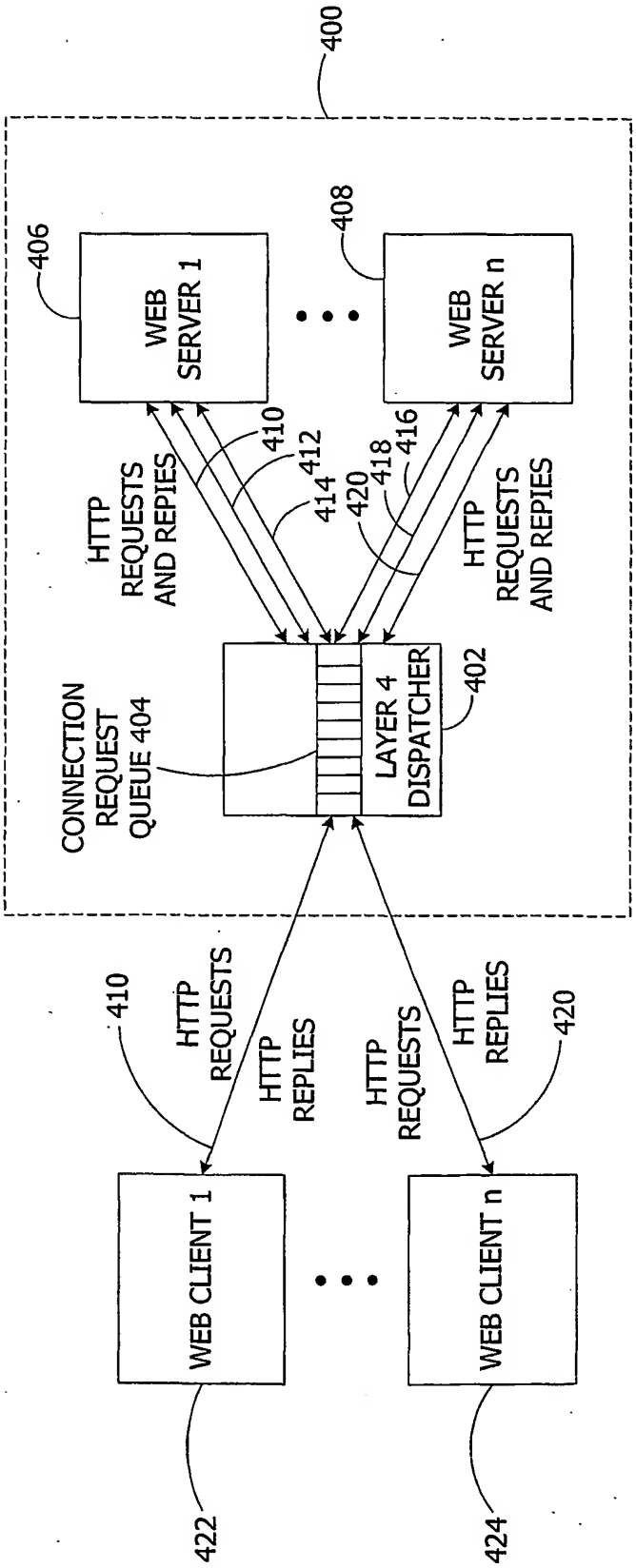

400
406
WEB SERVER 1
408
WEB SERVER n
HTTP REQUESTS AND REPIES
410
412
414
420
418
416
HTTP REQUESTS AND REPIES
CONNECTION REQUEST QUEUE 404
LAYER 4 DISPATCHER
402
410
HTTP REQUESTS
HTTP REPLIES
HTTP REQUESTS
HTTP REPLIES
420
WEB CLIENT 1
WEB CLIENT n
422
424

FIG. 4

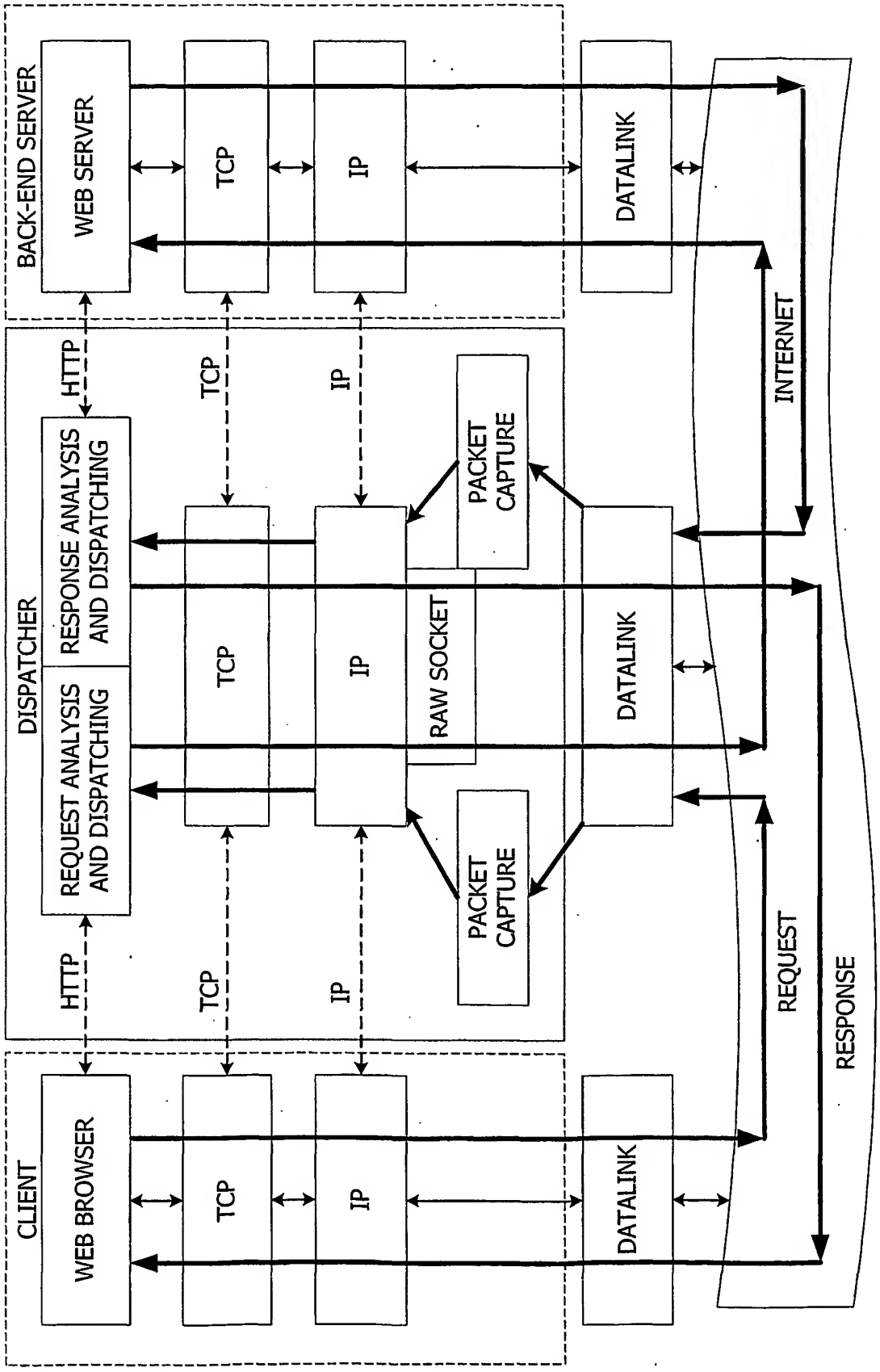

BACK-END SERVER
WEB SERVER
TCP
IP
DATALINK
INTERNET
DISPATCHER
HTTP
RESPONSE ANALYSIS AND DISPATCHING
REQUEST ANALYSIS AND DISPATCHING
TCP
IP
PACKET CAPTURE
RAW SOCKET
DATALINK
PACKET CAPTURE
HTTP
TCP
IP
CLIENT
WEB BROWSER
TCP
IP
DATALINK
REQUEST
RESPONSE

FIG. 5

FIG. 6

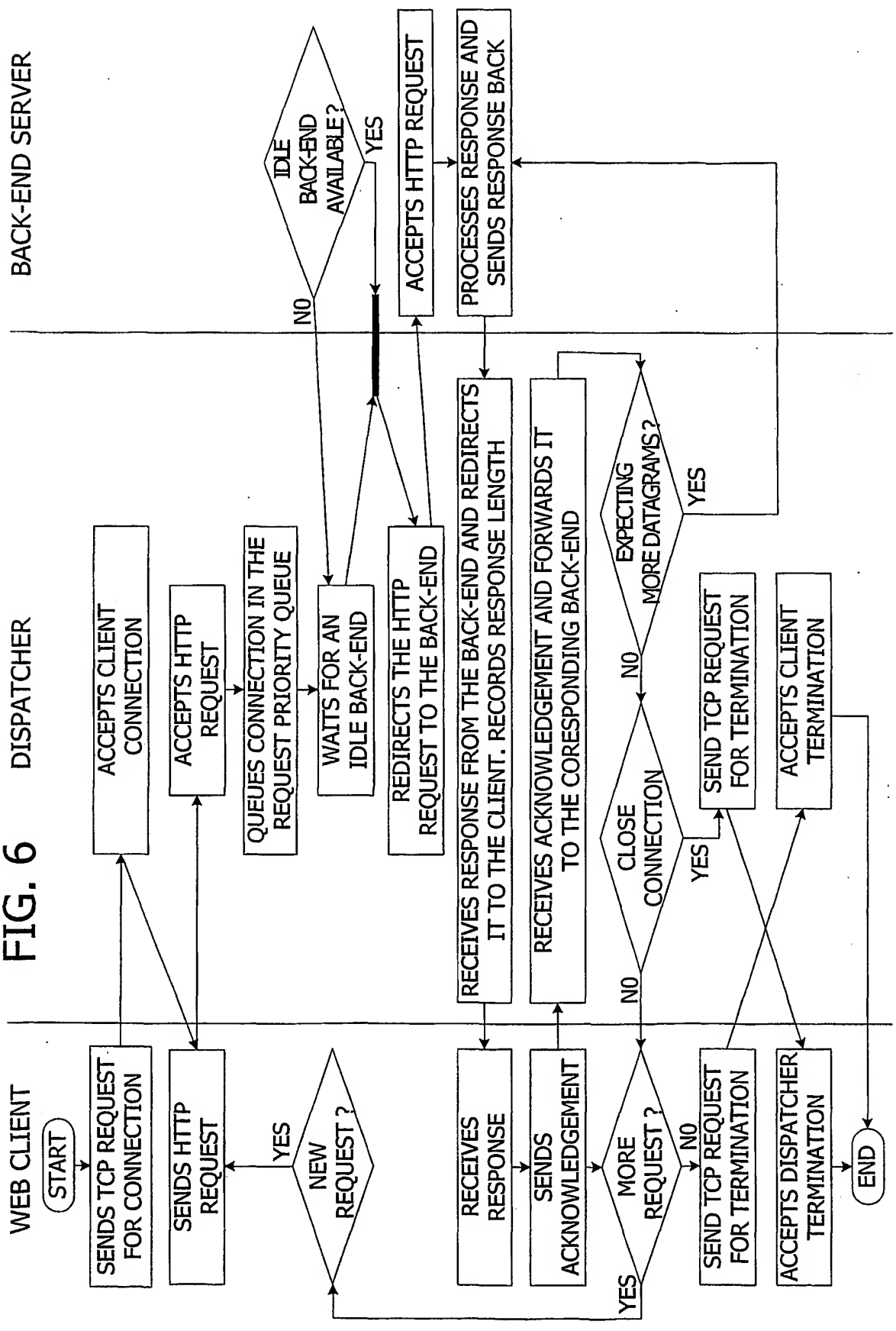

WEB CLIENT
DISPATCHER
BACK-END SERVER
START
SENDS TCP REQUEST FOR CONNECTION
ACCEPTS CLIENT CONNECTION
SENDS HTTP REQUEST
ACCEPTS HTTP REQUEST
QUEUES CONNECTION IN THE REQUEST PRIORITY QUEUE
WAITS FOR AN IDLE BACK-END
IDLE BACK-END AVAILABLE?
YES
NO
REDIRECTS THE HTTP REQUEST TO THE BACK-END
ACCEPTS HTTP REQUEST
PROCESSES RESPONSE AND SENDS RESPONSE BACK
NEW REQUEST ?
YES
RECEIVES RESPONSE
RECEIVES RESPONSE FROM THE BACK-END AND REDIRECTS IT TO THE CLIENT. RECORDS RESPONSE LENGTH
SENDS ACKNOWLEDGEMENT
RECEIVES ACKNOWLEDGEMENT AND FORWARDS IT TO THE CORESPONDING BACK-END
MORE REQUEST ?
YES
NO
EXPECTING MORE DATAGRAMS?
YES
NO
CLOSE CONNECTION
YES
NO
SEND TCP REQUEST FOR TERMINATION
SEND TCP REQUEST FOR TERMINATION
ACCEPTS CLIENT TERMINATION
ACCEPTS DISPATCHER TERMINATION
END

FIG. 7B

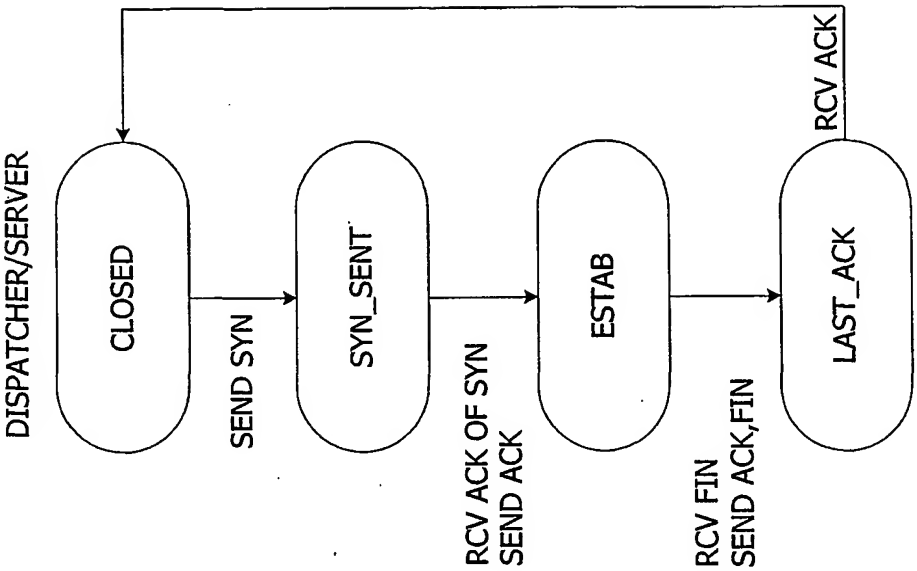

DISPATCHER/SERVER
CLOSED
SEND SYN
SYN_SENT
RCV ACK OF SYN
SEND ACK
ESTAB
RCV FIN
SEND ACK,FIN
LAST_ACK
RCV ACK

FIG. 7A

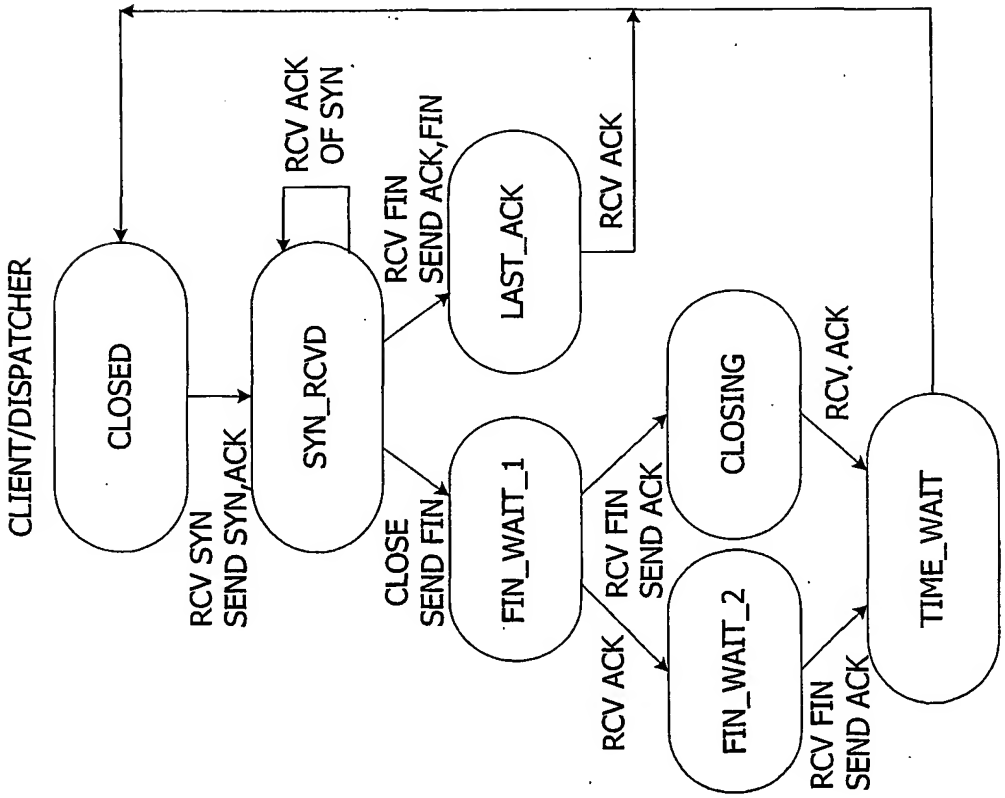
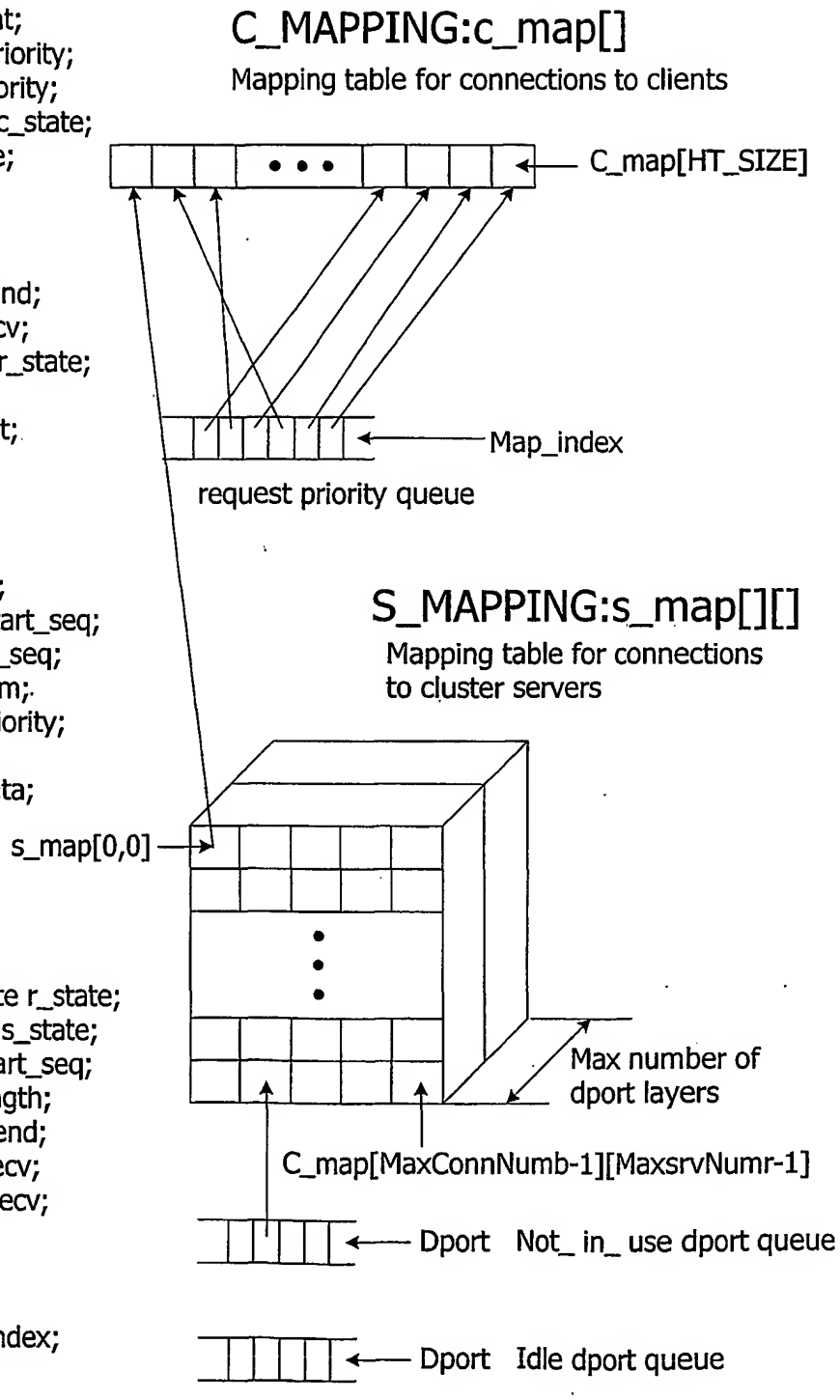

CLIENT/DISPATCHER
CLOSED
RCV SYN
SEND SYN,ACK
SYN_RCVD
RCV ACK
OF SYN
CLOSE
SEND FIN
FIN_WAIT_1
RCV FIN
SEND ACK,FIN
LAST_ACK
RCV ACK
RCV ACK
RCV FIN
SEND ACK
CLOSING
RCV ACK
RCV FIN
SEND ACK
FIN_WAIT_2
TIME_WAIT

FIG. 8

```
typedef structure{
    uint8_t flags;
    uint32_t collcount;
    uint16_t conn_priority;
    float domain_priority;
    enum net_state c_state;
    time_t start_time;
    uint32_t clP;
    uint16_t cport;
    uint16_t dport;
    uint32_t next_send;
    uint32_t next-recv;
    enum req_state r_state;
    req   *request;
    bool first_request;
}c_mapping;

typedef structure{
    bool conn_close;
    uint32_t dtoc_start_seq;
    uint32_t c_start_seq;
    uint16_t req_num;
    float req_file_priority;
    int req_size;
    u_char  *req_data;
    req *next;
}req;

typedef struct{
    enum dport_state r_state;
    enum net_state s_state;
    unt32_t serv_start_seq;
    int response_length;
    uint32_t next_send;
    uint32_t next_recv;
    uint32_t temp_recv;
    uint32_t slP;
    uint16_t sport;
    uint16_t dport;
    uint32_t hash_index;
}s_mapping;
```



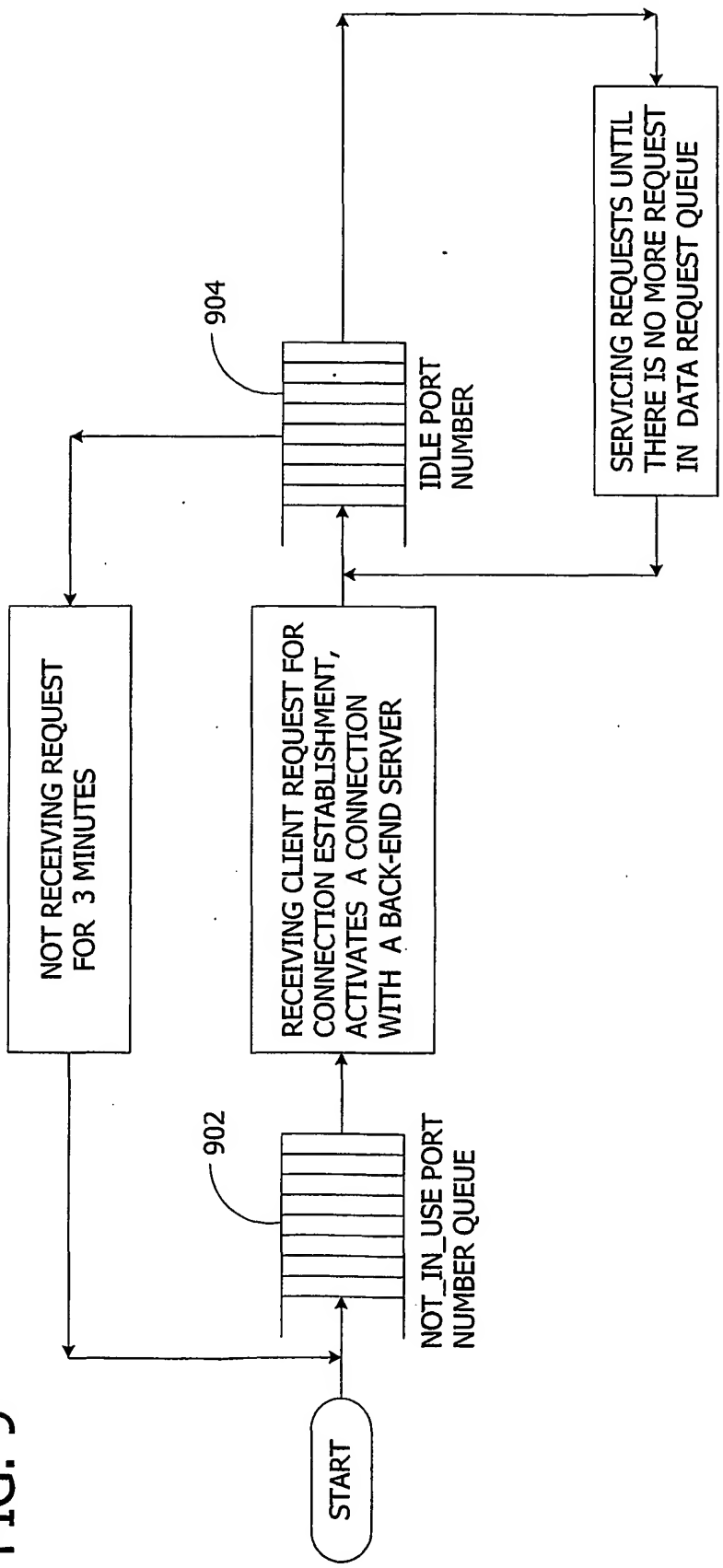
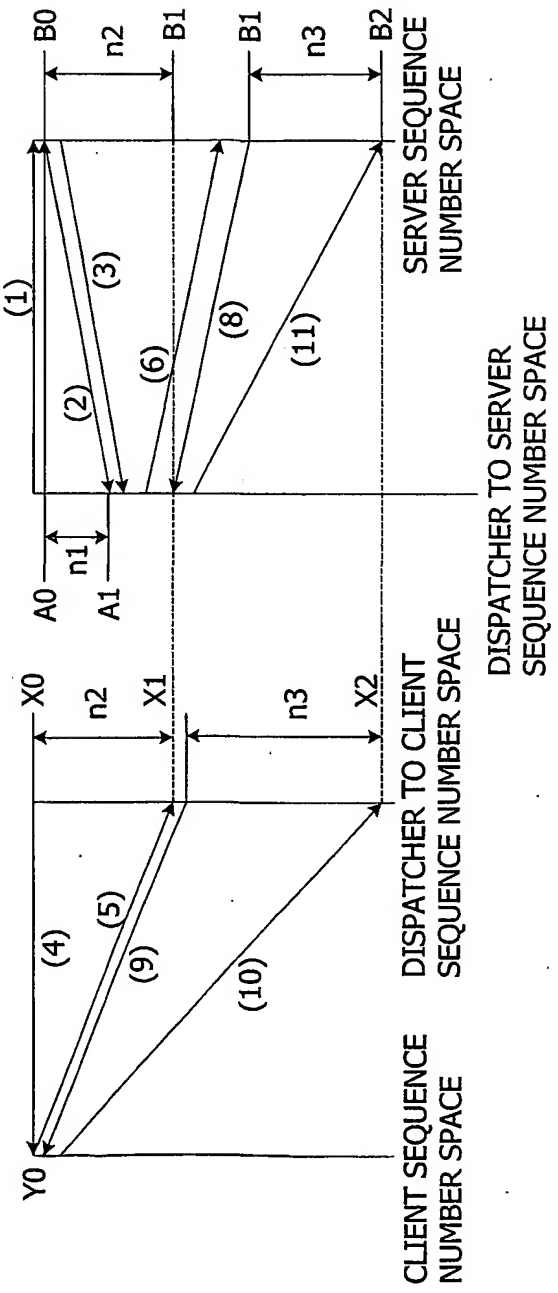

START
NOT_IN_USE PORT NUMBER QUEUE
902
RECEIVING CLIENT REQUEST FOR CONNECTION ESTABLISHMENT, ACTIVATES A CONNECTION WITH A BACK-END SERVER
NOT RECEIVING REQUEST FOR 3 MINUTES
IDLE PORT NUMBER
904
SERVICING REQUESTS UNTIL THERE IS NO MORE REQUEST IN DATA REQUEST QUEUE

FIG. 9

FIG. 10


B0
n2
B1
B1
n3
B2
SERVER SEQUENCE NUMBER SPACE
(1)
(2)
(3)
(6)
(8)
(11)
DISPATCHER TO SERVER SEQUENCE NUMBER SPACE
A0
n1
A1
X0
n2
X1
n3
X2
DISPATCHER TO CLIENT SEQUENCE NUMBER SPACE
(4)
(5)
(9)
(10)
Y0
CLIENT SEQUENCE NUMBER SPACE

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
10 May 2002 (10.05.2002)

PCT

(10) International Publication Number
WO 02/037799 A3

(51) International Patent Classification[7]: H04L 29/06, G06F 9/46

(21) International Application Number: PCT/US01/47013

(22) International Filing Date:
5 November 2001 (05.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/245,788	3 November 2000 (03.11.2000)	US
60/245,789	3 November 2000 (03.11.2000)	US
60/245,790	3 November 2000 (03.11.2000)	US
60/245,859	3 November 2000 (03.11.2000)	US
09/878,787	11 June 2001 (11.06.2001)	US
09/930,014	15 August 2001 (15.08.2001)	US
09/965,526	26 September 2001 (26.09.2001)	US

(71) Applicant: THE BOARD OF REGENTS OF THE UNIVERSITY OF NEBRASKA [US/US]; 3835 Holdrege, Lincoln, NE 68588-0745 (US).

(72) Inventor: GODDARD, Steve; The Board of Regents of the University of Nebraska, 3835 Holdrege Street, Lincoln, NE 68588-0745 (US).

(74) Agents: THOMAS, Michael, J. et al.; Harness, Dickey & Pierce, 7700 Bonhomme Avenue, Suite 400, St, Louis, MO 63105 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

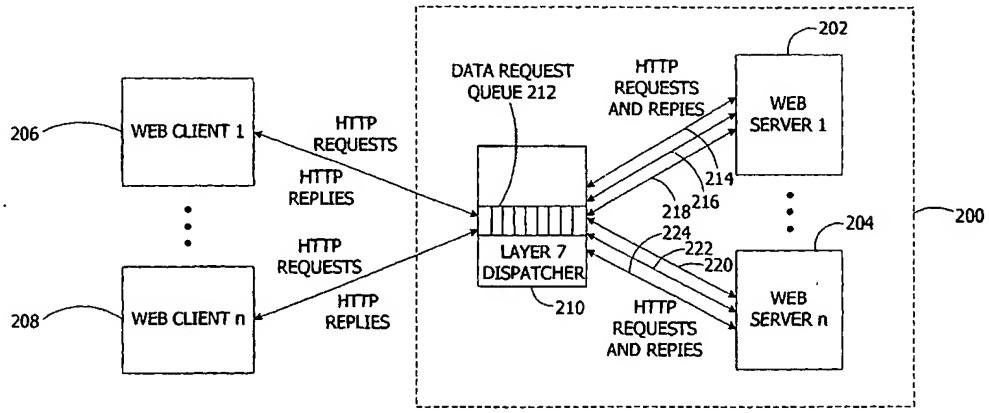
(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *with international search report*

(88) Date of publication of the international search report:
13 March 2003

[Continued on next page]

(54) Title: LOAD BALANCING METHOD AND SYSTEM



(57) Abstract: Standalone and cluster-based servers, including Web servers, control the amount of data processed concurrently by such servers to thereby control server operating performance. A dispatcher is preferably interposed between clients and one or more back-end servers, and preferably monitors the performance of each back-end server (either directly or otherwise). For each back-end server, the dispatcher preferably also controls, in response to the monitored performance, either or both the number of concurrently processed data requests and the number of concurrently supported connections to thereby control the back-end servers' performance. In one embodiment, the dispatcher uses a packet capture library for capturing packets at OSI layer 2 and implements a simplified TCP/IP protocol in user-space (vs. kernel space) to reduce data copying. Commercially off-the-shelf (COTS) hardware and operating system software are preferably employed to take advantage of their price-to-performance ratio.

WO 02/037799 A3

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

Inte ial Application No
PCT/US 01/47013

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L29/06 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 H04L G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, PAJ, WPI Data, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 6 141 759 A (BRADDY RICKY GENE) 31 October 2000 (2000-10-31)	1,3-6, 10, 19-26, 32,33, 39, 41-53, 60,61
Y	abstract column 9, line 43 - line 65 column 10, line 62 -column 11, line 17 column 19, line 30 - line 51 claims 1-3	2,7-9, 29-31, 34,40, 55,56,62

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
27 June 2002	23. 10. 2002

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Bertolissi, E

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Int ial Application No
PCT/US 01/47013

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X Y	EP 0 794 490 A (IBM) 10 September 1997 (1997-09-10) abstract page 1, line 52 -page 2, line 30 page 19, line 33 -page 22, line 23	11-13, 17,27,54 2,14,15, 18, 28-31, 35-37, 40,55, 56,63,64
Y	EP 0 892 531 A (SUN MICROSYSTEMS INC) 20 January 1999 (1999-01-20) abstract column 2, line 37 - line 56 column 3, line 37 -column 5, line 14 column 7, line 24 - line 49	14,15, 18,28, 35-37, 63,64
Y	ARIEL COHEN, SAMPATH RANGARAJAN, AND HAMILTON SLYE: SECOND USENIX SYMPOSIUM ON INTERNET TECHNOLOGIES AND SYSTEMS, 11 - 14 October 1999, XP002203108 Boulder, Colorado Abstract Introduction	7-9,34, 62
A	EP 1 035 703 A (LUCENT TECHNOLOGIES INC) 13 September 2000 (2000-09-13) abstract paragraph '0006! - paragraph '0007!	1-66
A	HUNT G D H ET AL: "Network Dispatcher: a connection router for scalable Internet services" COMPUTER NETWORKS AND ISDN SYSTEMS, NORTH HOLLAND PUBLISHING. AMSTERDAM, NL, vol. 30, no. 1-7, 1 April 1998 (1998-04-01), pages 347-357, XP004121412 ISSN: 0169-7552 Abstract 3. Managing TCP load allocation	1-66
A	WO 99 53415 A (WOLFF JAMES J ;HEWLETT PACKARD CO (US)) 21 October 1999 (1999-10-21) page 2, line 5 - line 12	1-66
A	EP 0 491 367 A (BULL HN INFORMATION SYST) 24 June 1992 (1992-06-24) abstract page 1, line 35 -page 2, line 18	1-66

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

national application No.
PCT/US 01/47013

Box I Observations where certain claims were found unsearchable (Continuation of item 1 of first sheet)

This International Search Report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☐ Claims Nos.:
because they relate to parts of the International Application that do not comply with the prescribed requirements to such an extent that no meaningful International Search can be carried out, specifically:

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box II Observations where unity of invention is lacking (Continuation of item 2 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

see additional sheet

1. ☐ As all required additional search fees were timely paid by the applicant, this International Search Report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fee, this Authority did not invite payment of any additional fee.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this International Search Report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☒ No required additional search fees were timely paid by the applicant. Consequently, this International Search Report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

1-66

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

This International Searching Authority found multiple (groups of) inventions in this international application, as follows:

1. Claims: 1-66

   Server, method and computer readable medium for dynamically adjusting the number of concurrent connections that a server is permitted to support

2. Claims: 67-84

   Server and method for reducing connection overhead

INTERNATIONAL SEARCH REPORT

Information on patent family members

Int al Application No
PCT/US 01/47013

Patent document cited in search report		Publication date	Patent family member(s)		Publication date
US 6141759	A	31-10-2000	US	6304967 B1	16-10-2001
EP 0794490	A	10-09-1997	US	6182109 B1	30-01-2001
			EP	0794490 A2	10-09-1997
			JP	3229237 B2	19-11-2001
			JP	9265409 A	07-10-1997
			KR	253930 B1	15-04-2000
EP 0892531	A	20-01-1999	US	6263368 B1	17-07-2001
			CA	2241016 A1	19-12-1998
			EP	0892531 A2	20-01-1999
			JP	11143804 A	28-05-1999
EP 1035703	A	13-09-2000	US	6314465 B1	06-11-2001
			DE	60000172 D1	27-06-2002
			EP	1035703 A1	13-09-2000
WO 9953415	A	21-10-1999	US	6101508 A	08-08-2000
			US	6067545 A	23-05-2000
			US	6185601 B1	06-02-2001
			US	6044367 A	28-03-2000
			AU	3861399 A	01-11-1999
			WO	9953415 A1	21-10-1999
EP 0491367	A	24-06-1992	US	5278984 A	11-01-1994
			CA	2058022 A1	20-06-1992
			DE	69123334 D1	09-01-1997
			DE	69123334 T2	26-06-1997
			EP	0491367 A2	24-06-1992

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☑ FADED TEXT OR DRAWING
- ☑ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.